CHAPTER V

CONTROL ARCHITECTURE

5.1 Proposed Architecture

The controller architecture was designed with the goal of providing a nonlinear learning controller with stability assurances. Central to the controller design process was the feasibility for actual field use. Care was taken not to make simplifying assumptions that are violated in actual applications. Previous work in adaptive controls (Wittenmark and Astrom, 1984) and artificial neural networks (Rumelhart and McCelland, 1986) highlighted two important points deemed necessary to successful adaptive control application: incorporation of *a priori* knowledge and assurance of stability.

To address these issues, the proposed control architecture uses *a priori* knowledge to assure system stability. The approach is pragmatic and is integrated into the overall plant design process. The control architecture is illustrated in Figure 5.1.

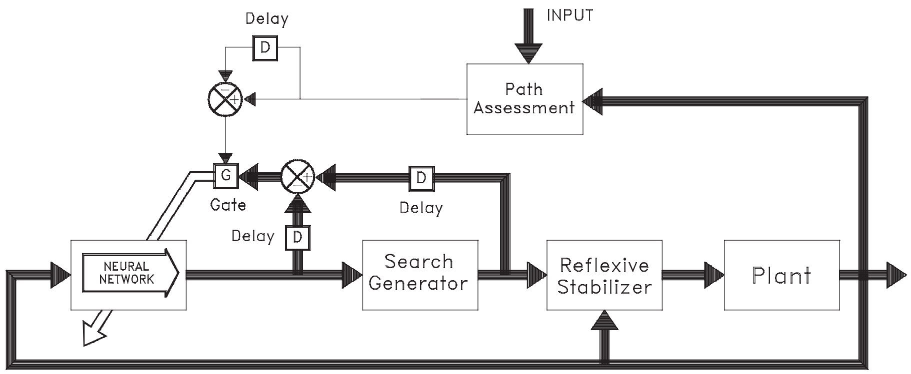

Figure 5.1: Control architecture

The basic controller consists of four components: an artificial neural network controller, a search generator, a path assessor, and a reflexive stabilizer. Without fitting a model to the system, the control architecture assures system stability while providing adaptive control of difficult nonlinear systems. The reflexive stabilizer is the primary component that directly incorporates *a priori* knowledge. Its purpose is to assure stability of the system by bounding the operating domain of the system. The neural network is an adaptive controller that provides control within the boundaries set by the reflexive stabilizer. This controller design separates system constraints from the goals of the adaptive controller.

The search generator and path assessor also require *a priori* plant knowledge, but to a lesser degree than does the reflexive stabilizer. The search generator explores the operating domain. The path assessor, a type of non-adaptive critic, guides the neural network in learning to build an attractor at the desired location. Together, the path assessor and the search generator instruct the network in building an attractor.

This chapter describes each component of the control architecture and presents the distal training algorithm. Section 5.2 explains the function of a reflexive stabilizer and describes how it forms an active Lyapunov bound of the plant. The design of the reflexive stabilizer is the topic of Section 5.3. The artificial neural network controller is described in Section 5.4. The distal-error training algorithm, an alternative method for training the neural network controller, is the topic of Section 5.5. Section 5.6 presents the path assessor and explains how it constructs an attractor. The final section, Section 5.7, presents the search generator.

5.2 Reflexive Stabilizer

The primary responsibility of the reflexive stabilizer is to assure stability of the plant by direct preemptive control when the plant nears an operational boundary. The reflexive stabilizer also may perform a secondary function of training the neural network. As presently implemented, the reflexive stabilizer is similar to an expert system (Astrom, 1986). It is built from a set of rules governing the operation of the system when a boundary is reached. The reflexive stabilizer boundaries are set within the actual failure boundaries of the system. The distance between the reflexive stabilizer boundaries and the system failure boundaries provides time and state space for the reflexive stabilizer to bring the system back into the reflexive stabilizer boundaries.

Each rule is designed to control the system so that it returns the system back inside the boundaries. The rules are only activated if a boundary is violated. If the system is within the specified boundaries, then the reflexive stabilizer is passive and the neural network controller controls the system. Thus, the reflexive stabilizer forms an active Lyapunov constraint for the system. This contrasts with the common use of Lyapunov analysis to prove that the controller and plant together are constrained in their operation. The idea is to augment the plant system with a controller that has a known Lyapunov bound. Instead of finding the Lyapunov bound for the system, a Lyapunov bound is placed where desired.

This design allows the stabilizer to concentrate on stability constraints, while the neural network or other adaptive controller can emphasize providing good control, where good depends on the application. The split in functionality also allows the controller components to be designed at different times. To understand the importance of this, the plant design and operation process need to be considered. Normally, a plant is designed by a team of engineers and scientists using experience from previous plants and design equations based on the underlying physics of the plant. After the plant is designed and built, a separate team is responsible for operating the plant. It is important to note that the design team has a general idea of how the plant will operate, but the fine details of operation are not known. The goal of the design team is to design a safe flexible plant. The operations team goal is to operate the plant at minimum cost. The minimum cost operation requires the operations team to capitalize on the fine details of plant operation in the face of variations in the plant and the operation environment.

An example is the heating and ventilation of a building. A designer will incorporate sufficient capacity and distribution equipment to accommodate a wide range of conditions. Then, the building operator will adjust flow rates and thermostat settings to yield the best results. The definition of best result is also subject to change. In the last example, the goal of the operator may change from best comfort to minimum energy or a multitude of other goals.

The reflexive stabilizer is designed by the design team, and the details of the adaptive controller are determined by the operations team. The design team designs the reflexive stabilizer with the help of simulation models and other predictors of basic system operation while the operations team has complete freedom to tune the adaptive controller based on experience gained from plant operation. Thus, under reflexive stabilizer supervision, the adaptive controller can be modified at any time without concern for plant stability and safety.

The separation of fine tuning from stabilization solves an expert controls problem. The problem is that the best rules for controlling the plant can be written only when the fine details of operation are available. The fine details, however, are not available until the plant is running. Yet, the plant needs the rules to operate. This quandary is usually resolved by starting with an initial set of rules and then revising them after the plant is in operation (Bernard, 1988b). The reflexive stabilizer needs rules to prevent failure, but these rules are much simpler to write since they only concern the boundary conditions. The tuning and fine points of operation are left to the adaptive controller.

A reflexive stabilizer or oversight controller has been used in one very significant application, the control of a nuclear reactor. Bernard *et al.*, (1984) and Bernard (1988b) have designed a "reactivity constraint" controller that has been certified by the United States Nuclear Regulatory Commission for use on the MIT 5MW research reactor. This controller was approved to allow any experimental controller, operating under the supervision of the reactivity constraint controller, to be tested on the system.

The main difference between the reactivity constraint controller and a reflexive stabilizer is that the reactivity constraint bounds the control while the reflexive stabilizer bounds the state space. The reactivity constraint controller restricts the reactivity of the nuclear reactor so that the prompt neutrons (directly controllable) can counteract the delayed neutrons (not directly controllable) at any instant. In other words, it is assuring that sufficient controllable neutrons are available to be removed from the reactor to bring the reactor to steady state power at any time. Thus, the reactivity constraint assures that the nuclear reactor is maintained under control at all times. In contrast, the reflexive stabilizer is primarily concerned with bounding the state space. As long as the plant is within the specified state space boundaries, the reflexive stabilizer is passive. The reflexive stabilizer places no bounds on the control. The reactivity constraint controller assures that the system is always controllable while the reflexive stabilizer catches the plant when it is out of control.

The possible secondary function of the reflexive stabilizer is to train the adaptive controller. In this capacity, the reflexive stabilizer is a new method for training a neural network controller. While the reflexive stabilizer only trains the neural network when the boundaries are violated, it provides more information than is usual for a reinforcement learning. The reflexive stabilizer provides the neural network with a target output value instead of a simple scalar good/bad signal. Thus, the training lies halfway between supervised learning and reinforcement learning. Training a neural network with the distal-error training algorithm (Section 5.5) and a reflexive stabilizer is investigated in Section 7.3.

5.3 Reflexive Stabilizer Design

This section describes how a reflexive stabilizer is developed. Readers not interested in the details of reflexive stabilizer design can safely skip this section. The previous section, Section 5.2, contains the fundamental concepts necessary to understand the other sections in the dissertation.

For the reflexive stabilizer to form a Lyapunov bounding box for the system, all the state variables of the system must be actively bound. Therefore, the foundation of the reflexive stabilizer is the physics model of the system. The model does not need to be finely calibrated; it only needs to model the gross motions of the system.

To illustrate the reflexive stabilizer design, the cart/pole system will be used (Figure 5.2). The goal of this task is to balance an inverted pendulum on the back of a cart by pushing on the cart.

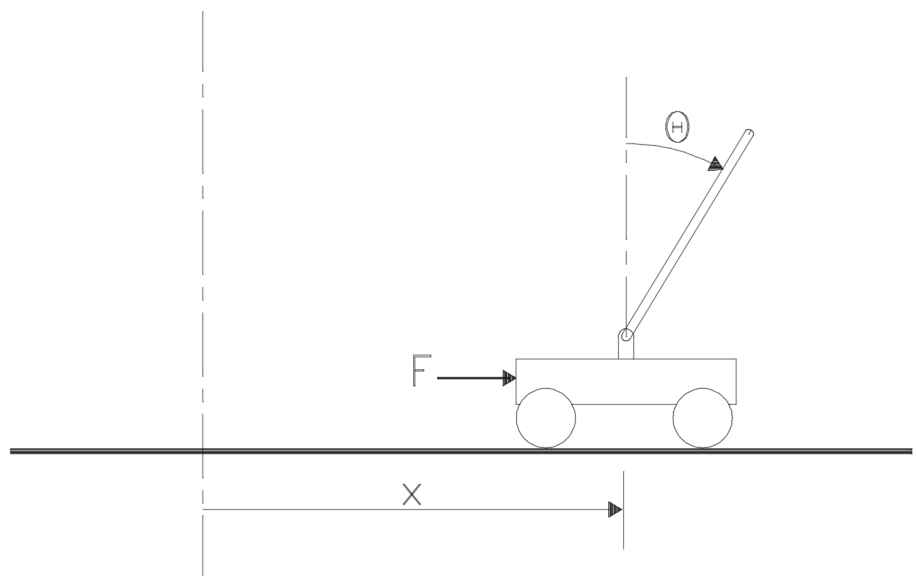

Figure 5.2: Cart-pole system

The equations of motion for this system according to Barto *et al.*, (1983) are:

$$\ddot{x}_t = \frac{F_t + ml\left[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t\right] - \mu_c\, sgn(\dot{x}_t)}{m_c + m}, \qquad \textbf{(5.2)}$$

$$\ddot{\theta}_t = \frac{g\sin\theta_t + \cos\theta_t\left[\dfrac{-F_t - ml\dot{\theta}^2\sin\theta_t + \mu_c sgn(\dot{x}_t)}{m_c + m}\right] - \dfrac{\mu_p\dot{\theta}_t}{ml}}{l\left[\dfrac{4}{3} - \dfrac{m\cos^2\theta_t}{m_c + m}\right]}, \tag{5.1}$$

where, g = acceleration of gravity,

m_c = 1.0 kg., mass of cart,

m = 0.1 kg., mass of pole,

l = 0.5 m, half-length of pole,

μ_c = varies, coefficient of friction for cart on track,

μ_p = varies, coefficient of friction for pole on cart,

F_t = 20 N, control force on cart, and

sgn(x) = -1.0 if x < 0, else sgn(x) = 1.0.

In the design of an actual system the above equations are probably available, but the exact values of both coefficients of friction are not known. Also unknown are the external disturbances and other possible un-modelled physical factors that will affect the system. Generally, ranges of these effects are estimated and the system is designed conservatively to handle the expected ranges.

Keeping this in mind, the reflexive stabilizer design begins with the determination of the fixed points (Percival and Richards, 1982) for the system. First, the equations 5.1 and 5.2 are converted to a system of first order differential equations:

$$\phi = \dot{\theta}, \tag{5.3}$$

$$y = \dot{x}, \tag{5.4}$$

$$\dot{\phi}_t = \frac{g \sin\theta_t + \cos\theta_t \left[\dfrac{-F_t - ml\phi^2 \sin\theta_t + \mu_c sgn(y_t)}{m_c + m} \right] - \dfrac{\mu_p \phi_t}{ml}}{l \left[\dfrac{4}{3} - \dfrac{m \cos^2\theta_t}{m_c + m} \right]}, \tag{5.5}$$

$$\dot{y}_t = \frac{F_t + ml \left[\phi_t^2 \sin\theta_t - \dot{\phi}_t \cos\theta_t \right] - \mu_c sgn(y_t)}{m_c + m}. \tag{5.6}$$

The fixed points are found by setting the derivative terms to zero and solving for the roots. The fixed point analysis for equations 5.3, 5.4 and 5.6 are not particularly helpful to the design of the reflexive stabilizer. The analysis of equation 5.5, however, results in the following equation:

$$0 = g\sin\theta_t - \frac{F\cos\theta_t}{m_c + m}, \tag{5.7}$$

which is a mathematical relation of the force required to balance the pole at given angle, θ. This is important because it indicates the maximum angle for which a given force on the cart can recover the pole. Figure 5.3 plots this relationship.

Given the above system constants, the 20 newton control force with no cart friction can balance the pole at a maximum angle of 61.6 degrees. Considering the unknown factors, this value was derated by a safety factor of approximately 2 to generate a 30 degree bound for θ. This was incorporated into the reflexive stabilizer with the following rule:

$$F = 20.0\, sgn(\theta)\, newtons \qquad |\theta| > 30^\circ . \tag{5.8}$$

The maximum available force of 20 newtons is used to move the cart/pole system as rapidly as possible back into the bounded region. A smaller force would increase the time for the reflexive stabilizer to return the system to within the desired bounds.

Once the maximum desired angle is set, the bounds on the pole's angular velocity may be determined. From Figure 5.3, it is apparent that to maintain an angle of 30 degrees slightly over 6 newtons of force is required.

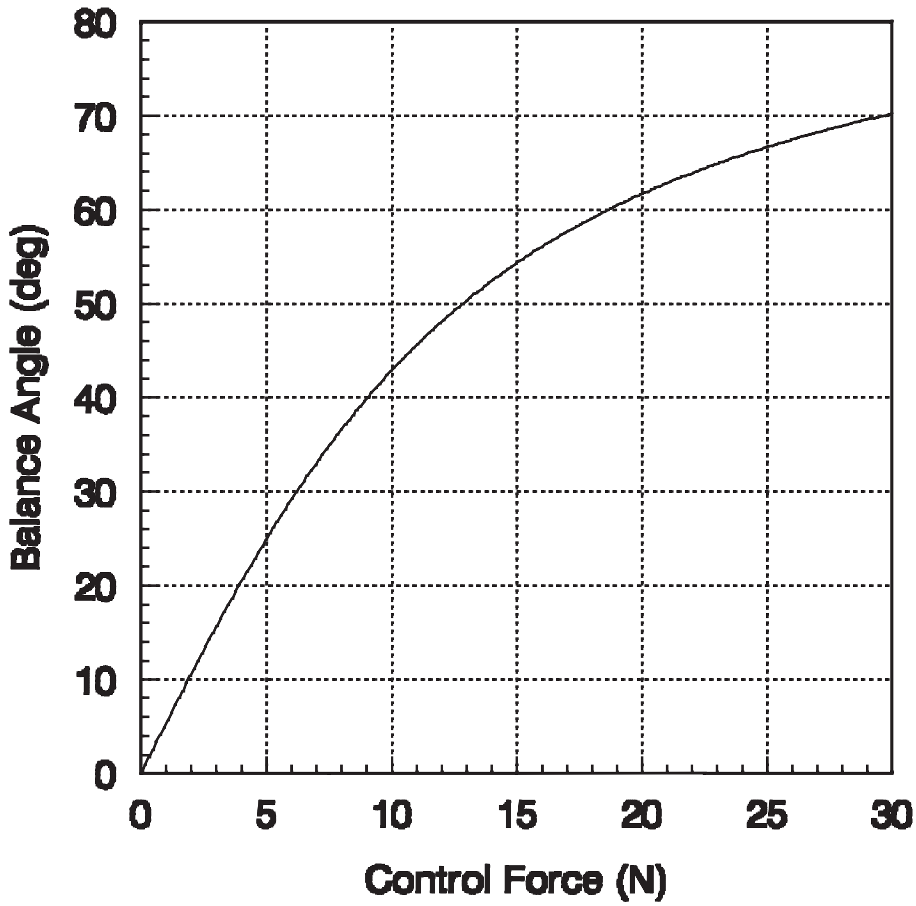

Figure 5.3: Pole balance angle as a function of control force. (Friction neglected)

This force is sufficient only if the pole has no positive angular velocity. If the pole has a positive angular velocity, then it will move past 30 degrees and require a greater force to balance the pole. Therefore, extra control force is required to arrest the angular velocity. Figure 5.4 illustrates this relationship.

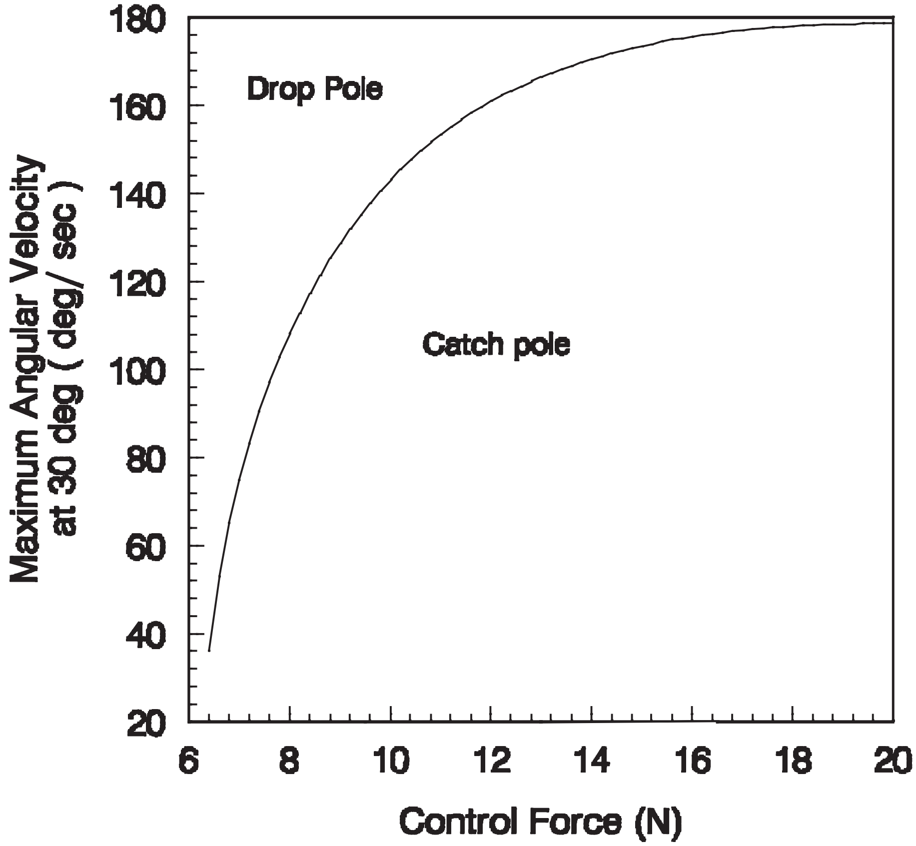

Figure 5.4: Control force versus maximum arrestable velocity as pole passes 30 degrees. (Friction neglected)

An available control action of 20 newtons can arrest the pole if it has less than 178 degrees/second of angular velocity at 30 degrees. This is the maximum angular velocity considering no friction or other disturbances. Therefore, the maximum angular velocity also must be held below this value to assure safe operation.

The reflexive stabilizer could be designed to allow higher angular velocities at angles within 30 degrees of vertical. It must be remembered, however, that the goal of the reflexive stabilizer is to provide stability and not to provide good control. Therefore, for sake of design simplicity, the velocity of the pole is bound with a single value.

The limit value, however, was selected because of its effect on the cart. The limit value must be less than 178 degrees/second, but high pole velocities result in high cart velocities. Therefore, in the interest of keeping the cart centered, a limit on pole velocities of 50 degrees/second was selected. This choice is a result of experience with the system via simulation studies. This constraint added the following rule:

$$F = 20.0\, sgn(\, \dot{\theta}\,)\, newtons \qquad |\dot{\theta}| > 50^{\circ}/\text{sec.} \tag{5.9}$$

The final two rules are designed to maintain the pole toward the center of the track. The first of these is to keep the velocity of the cart low and results in this rule:

$$F = 20.0\, sgn(\, \dot{x}\,)\, newtons \qquad |\dot{x}| > 1.0\ m/\text{sec.} \tag{5.10}$$

This purpose of this rule is subtle since it says to push in the direction of the excessive velocity when you want to reduce it. This results in the pole being moved to the side opposite the direction of the cart velocity, where a long term force, opposite the motion of the cart, can be applied. If a force were immediately applied opposite the motion, then the pole would move to the side of the motion and a force in the direction of the velocity would be needed to catch the pole.

The final pair of rules that are designed to restrain the cart position. The combination rule is:

$$\begin{aligned} F &= -20N \quad \textit{for} \quad x < -0.5 \quad \textit{and} \quad \theta < 10°, \\ F &= 20N \quad \textit{for} \quad x > 0.5 \quad \textit{and} \quad \theta < -10°. \end{aligned} \qquad \textbf{(5.11)}$$

This rule functions in a similar fashion to the previous rule by trying to keep the cart toward the center of the track by moving the pole to the side that the cart needs to move. The angle test is used to shut the rule off when the pole is on the correct side of the pivot.

The final design issue for the reflexive stabilizer is the precedence of the rules. The cart/pole system is somewhat pathological since there is a single control input to govern two degrees of freedom. The system is controllable as long as the time constant for each degree of freedom is different (Higdon, 1963), but the reflexive stabilizer has the unique situation of only acting to prevent failure. In this situation, the pole can be falling and the cart moving such that conflicting control action is required. The reflexive stabilizer is forced to make a decision. It was decided that dropping the pole was a larger failure than the cart moving away from the origin. Thus, the pole balance rules have precedence over the cart position rules. The resulting order of precedence in descending order is equation 5.9, 5.8, 5.10, and 5.11.

While reflexive stabilizer embodies significant plant operation knowledge, all the knowledge is general in nature. Fine details of plant operation are not required. Nevertheless, the general operation of the plant must be known to write an effective reflexive stabilizer. It is also important to remember that rules must be written for every state variable or the system may escape stability along an unguarded state.

In conclusion, the cart/pole reflexive stabilizer represents a difficult case. For many other systems, the limit rules are obvious. Plant limits on pressure, temperature, flow, and velocity are good sources of rules. The most difficult stabilizers to write are for inherently unstable systems and for systems where the plant dynamics need to be anticipated.

5.4 Artificial Neural Network Controller

The second major component of the control architecture is the adaptive controller (See Figure 5.1). This is the component that tunes itself to achieve the desired setpoint. An artificial neural network was selected for this component because of its ability to operate as an adaptive nonlinear controller that is capable of universal approximation and because it has a good training algorithm. To design a neural network controller, two primary issues must be addressed: selection of a learning method and state space exploration. A reinforcement learning method also adds the issue of temporal credit assignment. In this dissertation, two learning methods were used to train the controller: standard back-propagation (Section 3.6) and the distal error training algorithm (Section 5.5).

The primary motivation for the development of the reinforcement training algorithm is to accommodate training of the neural network by the reflexive stabilizer. The idea is to teach the network not to encounter the stabilizer. The resulting algorithm is a fixed memory algorithm with ties to back-propagation through time (Nguyen and Widrow, 1989; Werbos, 1990c) and the methods of Robinson and Fallside (1987). The mathematical details of the training algorithm are presented in Section 5.5.

The state space exploration is important mainly for on-line learning. Off-line methods simply use available data or they initially sample the operation domain randomly. On-line learning, however, is limited to the data from normal everyday operation of the plant that most likely is very local in nature. Unlike off-line training, a neural network trained on-line does not receive data samples covering the entire domain of operation.

The use of back-propagation training adds the additional problem of the network being a gaussian learner (i.e., it learns best what it sees the most). Controllers by their very nature spend most of the time in specific positions of state space. Thus, the network learns the local domain, but it will not learn the conditions when the system is far from the setpoint.

In off-line training, where all the data is available prior to training, the data histogram can be flattened by replicating data or adjusting the learning rate to derate the effect of common data. On-line training, however, is faced with the problem that future data is unknown. A local function like a radial basis function could be used to form a histogram on the fly, but high dimensional spaces may make this impractical.

The technique applied here is to learn only when a better result is obtained. Thus, learning takes place only under certain conditions. This limits learning if the network has already learned the proper control. For further discussion of state space exploration, see Section 5.7. Controlling the learning alleviates both the problem of the network being a gaussian learner and the problem of limited state space exploration.

5.5 Distal-Error Training Algorithm

This section develops the distal-error training algorithm, which accomplishes the second objective set forth in the introduction. The reinforcement training algorithm operates by performing a series of steps, after which the outcome of the sequence is assessed. The result of the assessment is used to correct the network so that under the same conditions a new sequence is produced that returns an improved outcome. Normally, a reinforcement training algorithm only receives a scalar indication of performance at the end of the series of steps. The reflexive stabilizer, however, provides a target force that should have occurred just prior to failure. This provides both derivative and magnitude information, which is significantly more informative than a scalar good/bad signal. Thus, this new training algorithm lies halfway between a reinforcement learning algorithm and supervised learning algorithm (For more discussion of supervised learning and reinforcement learning see Sections 3.5, 3.6, and 4.6.).

5.5.1 Notation

The derivation of the training algorithm requires some new mathematical structures and operators. A presentation of the notation and operators is necessary. The mathematical structures are multi-dimensional arrays. To indicate the dimensions, each variable is subscripted with indices to each dimension. For example, the input vector **P** in this notation is $\mathbf{P}_{1q}$. The subscripted 1 in the notation is to highlight the fact that the vector is a row vector. When needed to avoid confusion, initial subscripts on variables are promoted to regular size and are appended to the variable name. Thus, the matrix, $\mathbf{W}_2$, with m rows and n columns becomes $\mathbf{W2}_{mn}$. The dimensional subscripts contain a minimum of two elements. Therefore, a single character subscript is a label and two or more characters in a subscript are dimension indices. At all times, uppercase symbols indicate vectors, matrices and multi-dimensional arrays. Lower case symbols indicate scalars and scalar versions of uppercase symbols. An exception to this notation is the activation function that is always represented as **f** or **g**. The final note on notation is the use of a number 1 with an overline. This indicates a vector, matrix, or multi-dimensional array that contains a one in each element.

5.5.2 Operators

The development of the distal-error training algorithm requires three new operators to manipulate the structures. The first operator is an element multiply and is indicated by:

$$C_{nmq} = A_{nmq} \odot B_{nmq}\,, \tag{5.12}$$

which in scalar notation is:

$$c_{ijk} = a_{ijk}\, b_{ijk} \qquad \forall \quad \begin{array}{l} i = 1 \ldots m \\ j = 1 \ldots n \\ k = 1 \ldots q. \end{array} \tag{5.13}$$

The element multiply operator was defined in a three dimensional space, but it also applies to larger and smaller spaces.

The next two operators are the result of splitting a regular matrix multiply into two separate operations. Using the new operators a common matrix multiply is indicated by:

$$D_{mq} = A_{mn} B_{nq} = \langle A \otimes B \rangle^{n}_{mnq}\,, \tag{5.14}$$

where the partial multiply operator:

$$C_{mnq} = A_{mn} \otimes B_{nq}\,, \tag{5.15}$$

is defined in scalar notation as

$$c_{ijk} = a_{ij}\, b_{jk} \qquad \forall \quad \begin{array}{l} i = 1 \ldots m \\ j = 1 \ldots n \\ k = 1 \ldots q, \end{array} \tag{5.16}$$

and the compaction operator:

$$D_{mq} = \langle C_{mnq} \rangle^{n} = \langle C \rangle^{n}_{mnq}, \qquad (5.17)$$

is defined in scalar notation as

$$d_{ik} = \sum_{j=1}^{n} c_{ijk} \qquad \forall \quad \begin{array}{l} i = 1 \ldots m \\ k = 1 \ldots q. \end{array} \qquad (5.18)$$

The subscripts of the compaction operator indicate the number and size of each dimension of the multi-dimensional structure. The superscript indicates the dimensions over which the compaction sum is to be performed. The partial multiply performs the row/column element multiplication without the summation of the resulting products. The compaction operator takes the multidimensional structure from the partial multiply and performs the summation. Together they complete all the operations of a common matrix multiply.

Due to the postponement of the summation, each partial multiply increases the number of dimensions of the resulting structure by one. Thus,

$$A_{mn} B_{nq} C_{qr} = \langle A \otimes B \otimes C \rangle^{nq}_{mnqr} = \langle D \rangle^{nq}_{mnqr} = E_{mr}. \qquad (5.19)$$

Supporting theorems and corollaries are presented in Appendix A. With this introduction to the new operators, the development of the training algorithm can begin. Initially, standard matrix algebra will be used. The new structures and operators will be introduced as required.

5.5.3 Algorithm Derivation

The starting point for the neural network controller design is the training technique of back-propagation (See Section 3.6). The training algorithm developed here is designed for feedforward neural networks. Consider the three layer neural network in Figure 5.5.

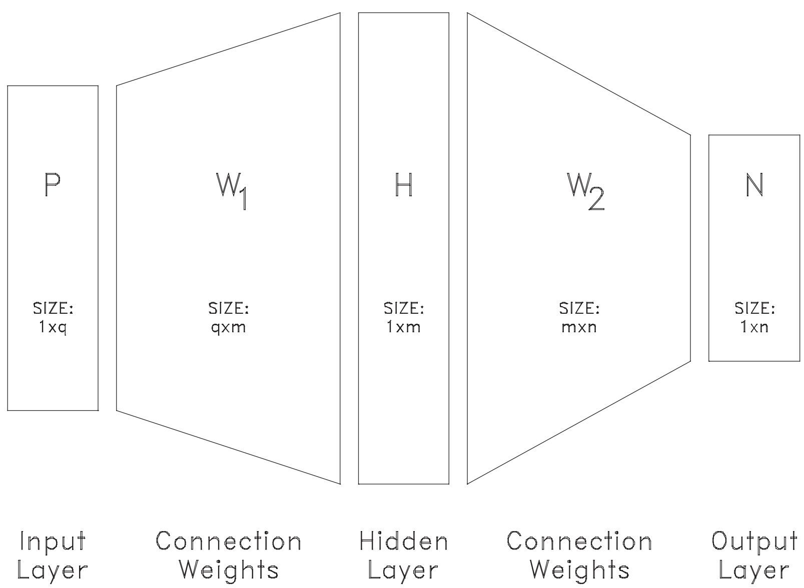

Figure 5.5: Three layer feedforward network.

The input to the network is the row vector **P,** which contains q elements; similarly the hidden layer, **H**, contains m elements and the output layer, **N**, contains n elements. The layers are connected with the weight matrices, $\mathbf{W_1}$ and $\mathbf{W_2}$. The forward activation of the network is:

$$H = f(P\,W_1), \qquad \textbf{(5.20)}$$

$$N = g(H\,W_2), \qquad \textbf{(5.21)}$$

where **f** and **g** are the requisite nonlinear functions.

The neural network is trained by comparing the network's output to the desired output and propagating the error back into the network to change the weights. The output error is defined as:

$$E = T - N, \qquad \textbf{(5.22)}$$

where **T** is the vector of desired network outputs.

Using this error function the update equation for the $\mathbf{W_2}$ connection layer is:

$$\Delta W_2 = H^T(g' \odot E), \qquad \textbf{(5.23)}$$

where **g'** is the derivative of the function **g** with respect to its independent variable.

The weight update equation for the hidden layer connections is the same except the error needs to be propagated through the $\mathbf{W_2}$ connection weights. In matrix notation this error is:

$$E_{hidden} = (g' \odot E) W_2^T. \tag{5.24}$$

Substituting this error into equation (5.23), the weight update equation for the $\mathbf{W_1}$ layer is:

$$\Delta W_1 = P^T \left[f' \odot \left\{ (g' \odot E) W_2^T \right\} \right]. \tag{5.25}$$

If a target value is available for every input pattern, then equations (5.23) and (5.25) are sufficient. The delta weights may be applied immediately or summed and then applied after a series of steps. Under reinforcement learning conditions, however, the target is only available after a series of input patterns are presented. The neural network must first perform a series of steps, then based on the final outcome, it adjusts its weights. This is illustrated in Figure 5.6.

Back-propagation through time solves the problem by positioning a neural network or other plant model between successive time steps (See Plant Model Controllers, Section 4.5). Then, the error can be propagated back through both the control and model networks to correct the network at earlier time steps (Nguyen and Widrow, 1989). The weight changes for each

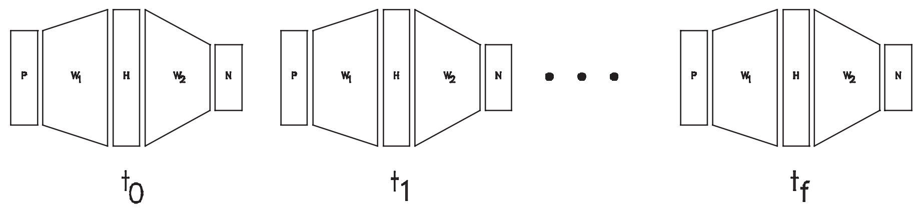

Figure 5.6: The temporal credit assignment problem. Error information available only at t_f.

time step are summed up and then the weights are adjusted. The drawback to this method is that the number of states saved increases with the length of the series. Thus, very long sequences can result in large memory requirements.

Reinforcement learning systems approach the temporal credit assignment from a different side. Rather than propagating the error through the network, the error is fed directly back to each time step. The physical system causality is then resolved through stochastic approximation. The idea is that the error from many different failures will, on the average, properly correct the network (Barto *et al.*, 1983; Sutton, 1988). To aid the process, a decay factor can be used to derate the effect of the error to steps early in the sequence. Thus, the connection weights are updated with the following weighted sum:

$$\Delta W_x = \sum_{t=t_0}^{t_f} \gamma^{t_f - t} W_x(t), \tag{5.26}$$

where $0 < \gamma < 1$, and x ranges over the connection weight layers.

Equation 5.26 is the starting point for the training algorithm. For output units, equation 5.23 is substituted into equation 5.26:

$$\Delta W_2 = \sum_{t=t_0}^{t_f} \gamma^{t_f - t} H^T(t)[g'(t) \odot E(t_f)]. \tag{5.27}$$

Then by application of Theorem 2 (See Appendix A) the update equation becomes:

$$\Delta W_2 = \sum_{t=t_0}^{t_f} \gamma^{t_f - t} (H^T(t) g'(t)) \odot (\bar{1} E(t_f)). \tag{5.28}$$

Noting that the error, **E**, is the same for all the terms and applying the distributive and commutative laws for the element multiply operator results in equation 5.29:

$$\Delta W_2 = (\bar{1} E(t_f)) \odot \sum_{t=t_0}^{t_f} \gamma^{t_f - t} H^T(t) g'(t). \tag{5.29}$$

This form of the update equation separates the error, which is available only at the end of the sequence, from the other terms that are distributed across the time series. This separation allows the summation to be computed incrementally, as

$$
\begin{aligned}
S(t_f) &= \sum_{t=t_0}^{t_f} \gamma^{t_f - t} H^T(t)\, g'(t) \\
&= H^T(t_f)\, g'(t_f) + \sum_{t=t_0}^{t_f - 1} \gamma^{t_f - t - 1} H^T(t)\, g'(t) \\
&= H^T(t_f)\, g'(t_f) + \gamma\, S(t_f - 1)\,.
\end{aligned}
\tag{5.30}
$$

Therefore, the final update equation for output layer connections (See Figure 5.5) is:

$$
\Delta W_2 = [\bar{1}\, E(t_f)] \odot S(t_f)\,. \tag{5.31}
$$

where **S(t$_f$)** is computed recursively as indicated by equation 5.30. This formulation stores all the past information in the fixed size data structure, **S**. The factor, γ, is a forgetting factor that controls the amount of past information retained and prevents summation overflow in the structure, **S**.

The hidden unit connections can be updated in a similar fashion, but due to the error passing, the update rule requires the new operators and structures. The development of the update rule for the Figure 5.5 hidden layer begins with substituting equation 5.25 into equation 5.26:

$$\Delta W_1 = \sum_{t=t_0}^{t_f} \gamma^{(t_f - t)} P^T(t) \left[f'(t) \odot \left\{ (g'(t) \odot E(t_f)) W_2^T(t) \right\} \right]. \quad \textbf{(5.32)}$$

The next step is to postpone the summation portion of the matrix multiply by using the partial multiply and compaction operators:

$$\Delta W_1 = \sum_{t=t_0}^{t_f} \gamma^{(t_f - t)} \left\langle P^T(t) \otimes \left[\hat{f}'(t) \odot \left\{ (g'(t) \odot E(t_f)) \otimes W2^T(t) \right\} \right] \right\rangle_{q1\,nm}^{1n}. \quad \textbf{(5.33)}$$

This corresponds to moving the summations to the beginning of the expression. In scalar notation this operation is:

$$\begin{aligned} \Delta W_1 &= \sum_{t=t_0}^{t_f} \gamma^{(t_f - t)} \sum_{k=1}^{1} p_{rk}(t)\, f'_{kj}(t) \sum_{i=1}^{n} g'_{ki}(t)\, e_{ki}(t_f)\, w2_{ij}(t) \\ &= \sum_{t=t_0}^{t_f} \sum_{k=1}^{1} \sum_{i=1}^{n} \gamma^{(t_f - t)} p_{rk}(t)\, f'_{kj}(t)\, g'_{ki}(t)\, e_{ki}(t_f)\, w2_{ij}(t) \qquad \textbf{(5.34)} \\ &\forall \quad j = 1 \ldots m \quad \textit{and} \quad r = 1 \ldots q. \end{aligned}$$

Noting that the summation over k is trivial, since the sum is from one to one, and noting that the error is independent of the time summation, equation 5.34 can be written as:

$$\Delta W_1 = \sum_{i=1}^{n} e_{ki}(t_f) \sum_{t=t_0}^{t_f} \gamma^{(t_f - t)}\, p_{rk}(t)\, f^{/}_{kj}(t)\, g^{/}_{ki}(t)\ w2_{ij}(t) \tag{5.35}$$

$$\forall \quad k = 1,\ j = 1 \ldots m \quad and \quad r = 1 \ldots q\,.$$

Thus, moving back to multidimensional notation and applying Theorem 1, the final update equation for the $\mathbf{W_1}$ layer (See Figure 5.5) is:

$$\begin{aligned} \Delta W_1 &= \left\langle\ (\overline{1_{q1}} \otimes E_{1n}(t_f) \otimes \overline{1_{nm}})\ \odot\ S_{q1\,nm}(t_f) \right\rangle^{1n} \\ &= \left\langle\ \hat{E}_{1n} \otimes S_{q1\,nm}(t_f) \right\rangle^{1n}, \end{aligned} \tag{5.36}$$

where,

$$S_{q1\,nm}(t_f) = \sum_{t=t_0}^{t_f} \gamma^{(t_f - t)}\, P^{\,T}(t) \otimes \left[\, f'(t) \odot \{\, g'(t) \otimes W2^{\,T}(t)\, \}\, \right]. \tag{5.37}$$

Equation 5.37 can be computed incrementally,

$$S_{q1\,nm}(t_f) = P^{\,T}(t) \otimes \left[\, f'(t) \odot \{\, g'(t) \otimes W2^{\,T}(t)\, \}\, \right] + \gamma\, S_{q1\,nm}(t_f - 1)\,. \tag{5.38}$$

This formulation is similar to Equation 5.30 where the history of a sequence is incrementally retained in a fixed size structure.

This formulation also has a major advantage when a critic or other external teacher is used to modify the weights in the middle of a sequence

(Sutton, 1988). If only the activations are retained, then the intra-sequence weight changes also affect the final outcome. Thus, at the end of the sequence, the effects of intra-sequence weight modifications and the consequences of the time series cannot be separated. The update equations, 5.31 and 5.37, however, include the current weights with the Jacobian matrices and eliminate this problem.

Equations 5.31 and 5.37 are the final weight update equations for the distal-error training algorithm. Together the equations form a new way for training neural networks. They also achieve the second primary objective. The use of the distal-error training algorithm for training with a reflexive stabilizer is explored in Section 7.3.

5.6 Path Assessment

Returning to Figure 5.1, the next component in the control architecture is the path assessor. The purpose of the path assessor is to build a fixed point attractor at the setpoint. It is the controller component that is necessary to accomplish the first primary objective. This section builds on the ideas presented in Section 4.6; a reader not familiar with reinforcement learning for control should review that section.

The path assessor is similar to a critic in design, but it does not estimate the future utility of the present action. Instead, it assesses the effect of the previous step and decides if the step should be remembered by the network. If the result of the step is good, then the path assessor opens the gate (see Figure 5.1) and allows the network to learn. Otherwise, the path assessor blocks learning. In other words, potential changes are stored in short term memory and used to correct the weights (long term memory) only if they return a good result. Otherwise new trial steps are saved to short term memory, and the old steps are overwritten and forgotten.

The path assessor monitors the effects of the search generator on the system. To illustrate the operation of the path assessor, consider Figure 5.7. The setpoint in state space is the point **C**. The vector, **P**, is the motion of the system through state space from **X'** at the previous sample point to **X** at the present point. The vector, **R**, reaches from the desired fixed point, **C**, to the present position, **X**. To form an attractor, the control forces on the system should work to turn the vector, **P**, toward the fixed point, **C**. The dot product of **R** with **P** should be negative. Therefore, the path assessor simply supplies the dot product between the two vectors. This dot product is delayed and compared to the dot product at the next sample step. If the result is negative, then the control force over the sample interval caused the

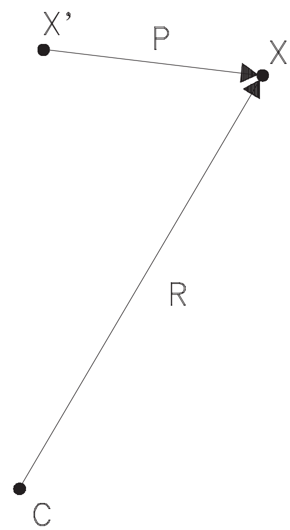

Figure 5.7: State space trajectory about desired fixed point.

path motion to move in an improved direction. A positive result indicates that the force moved the path away from the attractor. In response to this comparison, the gate in Figure 5.1 controls the network learning. A positive result prevents learning and negative result enables the network training.

The path assessor is the controller component that builds a fixed point attractor at the input setpoint. Its operation is studied using the underpowered pendulum system in Section 7.4.

5.7 Search Generator

The final component of the control architecture in Figure 5.1 is the search generator. The purpose of this component is to generate training examples from the plant. This component is necessary for on-line learning.

Exploring the state space is an important part of training a neural network. When a network is trained off-line, the training data can be hand selected to span the domain of operation. On-line training, however, requires a means for exploring the state space.

In the above control architecture, the search generator is a signal source that is added to the output of the neural network. The resulting sum is then truncated to the actuator range limits. The output of the search generator is the source of variations about the current network output. The combined signals are the test cases for the dynamic system. Each variation is appraised by the path assessor and the variations that yield improved performance are used as training examples for the network.

The most common method to search is to add a random variation to the control signal (Anderson, 1987; Barto *et al.*, 1982). The random variation method, however, has some limitations. The primary limitation is that it cannot supply a strong steady force, which is required in some situations. A good example of this type of situation is the swing up of the underpowered pendulum. To add energy to the pendulum, torque must be applied in the direction of rotation. A random variation search is unable to learn this, because the chance of many correct random forces all in the right direction is rare. When a random variation search is used with neural network that is a gaussian learner, the result is a local minimum in state space. The network learns to push the pole to one side or the other and oscillate about the maximum angle that the torque can support.

A search force is needed that varies at a rate that is longer than the period of the system. A good example of this would be an automobile throttle. If the throttle is jittered at a rapid rate, then the response of the engine will not follow the throttle. For the engine to follow the throttle, the throttle motion must be slow and deliberate. This effect is true, independent of the amplitude of the variations. The search generator used in the simulation tests of Sections 7.3 and 7.4 generated random gaussian noise. When a strong steady force was required, the reflexive stabilizer was used to solve the problem (See Sections 7.2 and 7.4).

The proposed control architecture with descriptions of each major component was presented in this chapter. The second objective, development of the distal-error training algorithm, was accomplished in Section 5.5. The control component necessary to accomplish the first objective, direct attractor construction, was presented in Sections 5.6. The entire control architecture is the foundation for the third objective. Both the first and third objectives are completed in Section 7.4.

CHAPTER VI

METHODOLOGY

6.1 Introduction

This chapter describes the physical systems studied, the hardware and software evaluation platform, the neural network architecture, and other practical implementation issues. Two unstable physical systems were selected. Each physical system was selected to facilitate the study of particular aspects of the proposed control architectures. The underpowered pendulum was selected as a low-dimensional problem that allows data visualization to be readily applied. The cart/pole problem was selected to demonstrate the capabilities of the reflexive stabilizer. It was also selected to allow comparison of the new techniques to the many other cart/pole studies in the literature.

Section 6.2 contains a description of the underpowered pendulum and presents the equation of motion. The cart/pole system is described in Section 6.3. The cart/pole system is identical with the system described in Section 5.3, but is presented again here for the reader's convenience.

Section 6.4 describes the hardware and software systems. The neural network architectures and activation function are the topic of Section 6.5. The final section, Section 6.6, discusses practical implementation issues.

6.2 Underpowered Pendulum

The underpowered pendulum, Figure 6.1, is a simple pendulum that is driven by a torque source at its pivot. The goal of the problem is to balance the pole at a particular angle, given any initial conditions. The difficulty is that the torque source is insufficient to turn the pendulum directly to the correct position. To accomplish the task, the pendulum must be swung back and forth until it can gain enough energy to reach the desired position.

The problem is highly nonlinear both in required control action and system dynamics. The controller must initially add energy to the system to get the pole to the vertical position. When the pole is near the upright position, it must damp the pendulum to a stop. Complicating the control is the inherent nonlinearity of the pendulum and friction at the pivot point.

The equation of motion for this system is:

where **l** is the pole length, **m** is the end mass, τ is the control torque, and **μ** is the coefficient of friction at the pivot point. For the pendulum to be

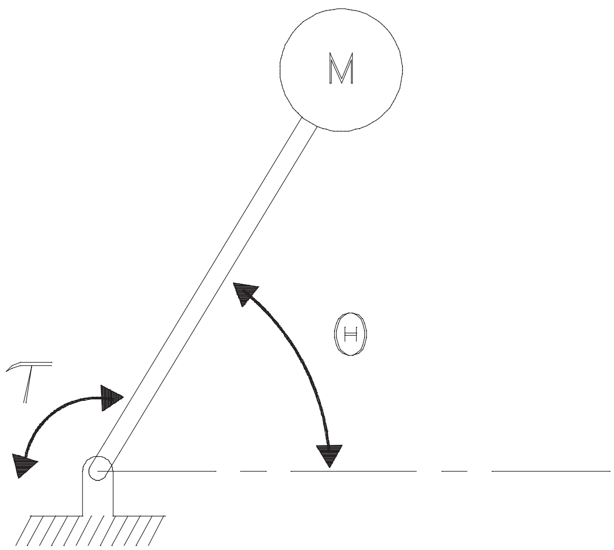

Figure 6.1: Underpowered pendulum

$$\ddot{\theta} = \frac{\tau - \mu}{ml} - g\cos\theta, \tag{6.1}$$

underpowered, the control torque, τ, must be insufficient to twist the pendulum from being stopped vertically down to the vertically upright position. Therefore,

$$\tau < mgl\cos\theta. \tag{6.2}$$

Given a pendulum with mass of 2 kilograms and a pole length of 1 meter, the torque must be less than 19.62 newton-meters. The maximum torque used in this dissertation was 12 newton-meters. At this torque level, the pendulum can be held within 52.3 degrees of vertical. Past 52.3 degrees of either side of vertical, the maximum torque is insufficient to overcome the force of gravity. The coefficient of friction in all underpowered pendulum tests was 0.2.

The underpowered pendulum was selected to demonstrate the attractor construction and the reflexive stabilizer. The system is of low dimension to facilitate data visualization, but it contains sufficient complexity to be of interest. Retained in the underpowered pendulum system are the difficult features of inherent instability and high nonlinearity.

6.3 Cart/Pole System

The cart/pole system is the other system under study (Figure 5.2). This system is identical with the example system presented in Section 5.3, but is presented here for convenience of the reader. The goal of this task is to balance an inverted pendulum on the back of a cart by pushing on the cart. The cart is also to remain as near the center of the track as possible.

Similar to the underpowered pendulum, the cart/pole system is highly nonlinear in both required control action and system dynamics. Although the pole operates over a smaller range than the underpowered pendulum, the addition of the cart dynamics to the pole dynamics results in a complex problem. Control of the system is complicated by competing control needs between the cart and pole and by the non-monotonic nature of the control. For example, if the pole is falling to the right and the cart is moving to the right, the proper control is first to push in the direction of the cart motion to move the pole to the side opposite the motion. Once the pole is on the side opposite the cart motion, a force may be applied to the left to arrest both the cart and pole motion. Thus, the cart's velocity must be increased before it can be reduced.

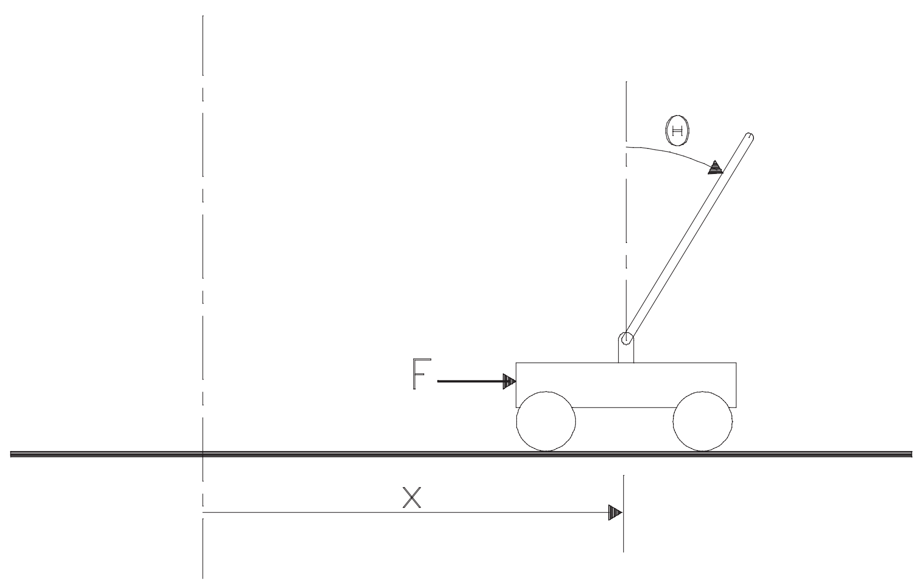

Figure 5.2: Cart-pole system

The equations of motion (Equation 5.1 and 5.2) for this system according to Barto *et al.,* (1983) are:

$$\ddot{\theta}_t = \frac{g\sin\theta_t + \cos\theta_t\left[\dfrac{-F_t - ml\dot{\theta}^2\sin\theta_t + \mu_c sgn(\dot{x}_t)}{m_c + m}\right] - \dfrac{\mu_p\dot{\theta}_t}{ml}}{l\left[\dfrac{4}{3} - \dfrac{m\cos^2\theta_t}{m_c + m}\right]}, \qquad \textbf{(5.1)}$$

$$\ddot{x}_t = \frac{F_t + ml\left[\dot{\theta}_t^2\sin\theta_t - \ddot{\theta}_t\cos\theta_t\right] - \mu_c sgn(\dot{x}_t)}{m_c + m}, \qquad \textbf{(5.2)}$$

where, g = acceleration of gravity,

m_c = 1.0 kg, mass of cart,

m = 0.1 kg, mass of pole,

l = 0.5 m, half-length of pole,

μ_c = 0.0005, coefficient of friction for cart on track,

μ_p = 0.000002, coefficient of friction for pole on cart,

F_t = 20 N, control force on cart, and

sgn(x) = -1.0, if x < 0, else sgn(x)= 1.0.

The problem was also complicated by adding a random disturbance to the position of the pole at each time step. The random gaussian disturbance has a range of 5 degrees. Therefore, at any instant the pole can be disturbed by as much as five degrees.

The cart/pole system was selected to demonstrate the distal-error training algorithm and the reflexive stabilizer. It represents a difficult control task and it is a problem that is common in the literature. The system contains the features of inherent instability and high nonlinearity that are found in many real physical systems.

6.4 Evaluation Platform

This section describes the computer systems and the software that were the foundation of the studies. The primary evaluation system for the simulation tests is an IBM PC compatible 25 Mhz i486 microcomputer. Three other 33 Mhz i386 microcomputers were also used on the computationally intense studies. The software platform was developed on System V.3 UNIX from Santa Cruz Operation. Most of the software was written in the computer language, C. Portions of the code were also written in i386 assembly language to gain speed in critical sections. Due to the binary compatibility of all computer systems, the code ran on both the i386 and i486 systems without modification.

All the neural network code was implemented using matrix formulations to allow easy optimizations of code. Some subroutines, like matrix multiplication, were hand coded in assembly language to gain computing efficiency. The assembly language coding increased the speed of the software by 2-4 times.

All the dynamic systems' equations of motion were solved with a 4th order Runge-Kutta integrator. The integration step size for both the underpowered pendulum and the cart/pole system was 0.01 seconds.

6.5 Neural Network Architecture

This section presents the details of the neural networks, which were used in the studies. The section begins by describing the structural details of the neural networks used in the simulation studies. The neural network details for the distal-error training and attractor construction tests are presented. The section concludes with a discussion of the activation function used in all the studies. The effects of the activation function on computational speed and neural network processing are discussed.

The inputs to each network were the state variables from the physical systems. The inputs were presented directly to the neural networks as continuous values. Each state variable was assigned an input node. No feature computations or other special representations were computed. Due to the common ranges of the state variables, the values were not scaled before presentation to the network. The only exception to this was the underpowered pendulum where the angular position was scaled to keep the value within $\pm\pi$. Additional discussion of input and output scaling is contained in Section 6.1.1

All the neural networks used in this dissertation were three layer feedforward networks. For the distal-error training tests, the neural network contained 4 input units, 10 hidden units, and 1 output unit. All units used an arc tangent activation function. Input to the network was the four cart/pole state variables. The network was trained with the learning rate set to 0.005 and the forgetting factor set to 0.01.

The attractor tests used a three layer neural network architecture with 2 input units, 15 hidden units and 1 output unit. Again, the arc tangent activation function was used for all the nodes. The inputs to the network were the angular position and angular velocity from the underpowered pendulum. The learning rate for the tests was set to 0.5.

A major difference from other neural network studies is the use of the arc tangent as the activation function. The primary motivation for this selection is the availability of the arc tangent in the microcomputer math co-processor. Thus, the arc tangent can be computed with a single instruction rather than the multiple instructions required to compute the other activation functions.

The arc tangent has two important features. First, the range of the function is $-\pi/2$ to $\pi/2$. The only effect of this is that the output must be scaled differently than the logistic or hyperbolic tangent. The second feature is that off the center axis the arc tangent decays to the minimum and maximum values more slowly than the other functions. This is illustrated in Figure 6.3.

The slower decay rate should help in learning. In the back-propagation update equation the error is multiplied by the derivative of the activation function. When the node's activation is large, the activation function derivative becomes small. This blocks error from reaching the node and makes changing the node's weights difficult. This phenomena is commonly referred to in the neural network literature as a unit becoming "railed".

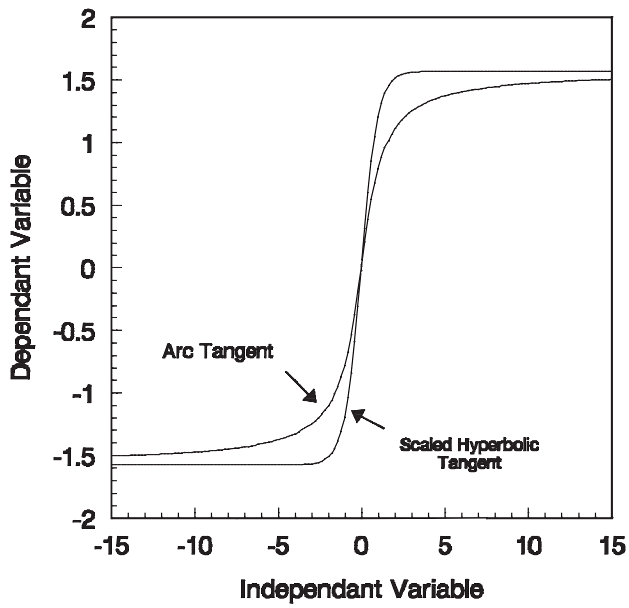

Figure 6.3: Comparison of hyperbolic tangent with arc tangent. (range of hyperbolic tangent scaled by $\pi/2$)

Since the arc tangent's derivative decays to zero more slowly than the hyperbolic tangent, more error is communicated to the network. This makes it more difficult for a node that is forming an incorrect representation to become stuck because the activation derivative allows more error to reach the node. In conclusion, the arc tangent provides improvements in computational speed and network learning.

6.6 Practical Issues

As in any engineering application, there are many practical implementation issues. This section focuses specifically on two issues: dynamic system measurement and neural network cycle times. Dynamic system measurement is the process of gathering information from the system. The rate of neural network processing relative to the physical system dynamics is the network cycle time.

6.6.1 Dynamic System Measurement

The first issue is to identify which measurements to gather from the physical system. From the work of Packard *et al.*, (1980) and Takens (1981) a single measured variable plus delayed versions of the variable are sufficient to reconstruct the state of the system[4]. The real issue is which measurements are the best ones to take. There are no solid answers to this problem, but there are several factors to aid the selection.

First, measurement of as many system states as possible is advisable. This saves the neural network from estimating them or their equivalent states. Generally, input information that is rich in the facts necessary to solve

---

[4]See Section 2.6 for further discussion of the Takens result.

the problem eases the job for the neural network (Heermann and Khazenie, 1992).

Second, data probably should be sampled according to the Nyquist sampling theorem, which is to take samples a minimum of twice as fast as the shortest period in the system. It may be possible to sample slower than the Nyquist frequency, but the problem is more difficult.

The final issue is data scaling. Strictly speaking, input data can be scaled by the first layer of weights. In practice, however, it is better to have all inputs set to similar ranges. This prevents one input from dominating another simply because its magnitude is greater. The output of the neural network must be scaled to the range of the actuators. This could be learned by the network with a linear activation function on the output layer, but both the neural network and actuator ranges are known *a priori*. Therefore, the neural network output can be directly scaled.

For all the studies in this dissertation, one of the neural network inputs and all the neural network outputs were scaled. The only scaled input was the angle of the underpowered pendulum. This was needed to keep multiple rotations of the pendulum within a 2π range. All the outputs were scaled so that the minimum and maximum network output produced the minimum and maximum actuator output.

6.6.2 Neural Network Cycle Times

The neural network cycle time is the speed of neural network operation with respect to the dynamic system. A neural network has three separate operations which can cycle at different frequencies. The separate operations are, 1) forward pass, 2) error calculation and backward pass, and 3) weight updates. The forward pass is the forward propagation through the neural network. This pass determines the actuator position for the measured input conditions. To maintain control of the system, the forward pass must occur fast enough to keep up with the system dynamics.

The other two network operations govern the learning. Commonly, the weight updates are performed immediately after each backward pass through the neural network. It is important that the network learn significantly slower than the system dynamics. This keeps the learning dynamics from interacting with the control of the system dynamics. Simultaneously, if the neural network is to be useful, then it must train in a reasonable time. If the network also can follow non-stationary effects in the system, it is beneficial.

For the simulations in this dissertation, all network cycles were synchronous with the integration time step. The one exception to the synchronous update is the distal-error algorithm where the error calculation and weight update occur only at the end of a sequence of actions.

Generally, the complete cycle consisted of three major steps: 1) A forward pass of the network was performed to generate a position for the actuator, 2) The system was integrated one time step with the Runge-Kutta integrator, and 3) The outcome of the physical system was assessed and the backward pass of the network was conducted if the assessment was positive. Preemptive at all times was the reflexive stabilizer. Any time the system encountered a reflexive stabilizer boundary, the reflexive stabilizer took control of the system.

The research methodology was to study the control architecture via computer simulation. Two inherently unstable physical systems, an underpowered pendulum and the cart/pole system, were selected to investigate the control techniques. Discussed in chapter 6 were details of the computer hardware and software. Details of the neural networks were also presented.

CHAPTER VII

SIMULATION RESULTS

7.1 Introduction

This chapter describes the results of the simulation tests. Section 7.2 describes the investigations of the reflexive stabilizers for both the cart/pole and underpowered pendulum. Section 7.3 contains the tests of the new training algorithm for learning with a distal error signal. The algorithm is demonstrated using the cart/pole system. The first objective, construction of an attractor to provide control, is demonstrated in Section 7.4. The fixed point attractor construction studies use the underpowered pendulum as the test system. Attractor building is demonstrated by balancing the pendulum vertically and 30 degrees off center.

7.2 Reflexive Stabilizer

Although the primary purpose of the reflexive stabilizer is to assure stability, it also can train the neural network, or act as an assistant controller to the neural network. Both the assistant controller and plant stabilization applications are the topics of this section. Training a neural network with a reflexive stabilizer is demonstrated in Section 7.3.

7.2.1 Reflexive Stabilizer as an Assistant Controller

To demonstrate the use of the reflexive stabilizer as an assistant controller, the pendulum system was used. For the underpowered pendulum problem no type of motion was considered a failure. The goal of the controller is to balance the pole at a particular angle when given any initial conditions.

To accomplish this task, two types of control are required. If the pendulum energy is insufficient to carry the pendulum into the upper domain, where the maximum torque is greater than the gravity forces, the controller must add energy to the system. Once there is sufficient energy to carry the pendulum into the upper controllable domain, then the controller must damp the pendulum and bring it to a stop at the desired angle.

Theoretically, a single neural network controller could accomplish both tasks. In practice, the upper and lower domains require different training methods. Also, due to the large control differences required, learning in each domain erases the knowledge from the other domain.

The knowledge is erased because the network spends time in lower domain learning to add energy to the system, then it moves to the upper domain where the control objective changes to damping the energy from the system. If single patterns from each domain were presented randomly, the neural network could learn both systems. The on-line learning, however, presents many patterns from one domain followed by many patterns from the other domain. Therefore, because the network continuously moves back and forth between domains, the network does not learn to solve either problem.

A solution is to divide the control between two controllers. Both controllers could be neural network controllers, but adding energy to the system is so simple that this task was assigned to the reflexive stabilizer. Damping and balancing was done with the neural network. To add energy to the underpowered pendulum, there is a single reflexive stabilizer rule: "If the energy is too low and the system is in the lower control domain, then apply torque in the direction of rotation." This rule swings the pendulum back and forth until sufficient energy is added to the system. There is sufficient energy in the pendulum when it has enough energy to swing up into the upper controller's domain of operation.

To illustrate this, consider the dynamics of pendulum system with a 0.2 friction coefficient. The uncontrolled dynamics of the pendulum are a fixed point attractor at an angle of 270 degrees with zero velocity. The phase space portrait of the motion is shown in Figure 7.1.

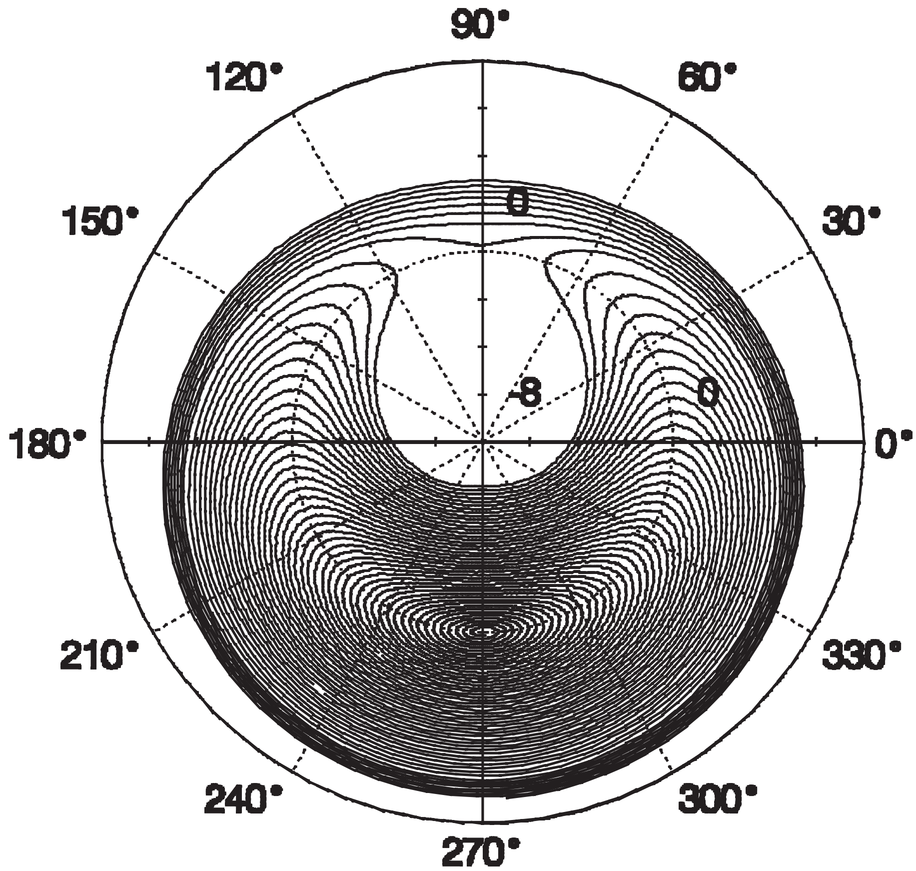

Figure 7.1: Phase plot of uncontrolled pendulum (μ=0.2).
(pole angle in degrees on theta axis, angular velocity in radians per second measured on radial axis)

In Figure 7.1, the pendulum is started with sufficient energy to rotate completely around the pivot. Friction at the pivot removes energy from the system; soon the pendulum can no longer make it over the top. The pendulum then swings from side to side, losing energy, until it comes to rest at 270 degrees with zero velocity. The attractor is a spiral where the pivot friction governs the rate of decay to the fixed point.

By augmenting the system with a reflexive stabilizer, the dynamics are modified to cancel the fixed point attractor. The combined system dynamics are a stable cycle. The reflexive stabilizer counteracts the natural fixed point attractor and forms the limit cycle shown in Figure 7.2. In Figure 7.2, the

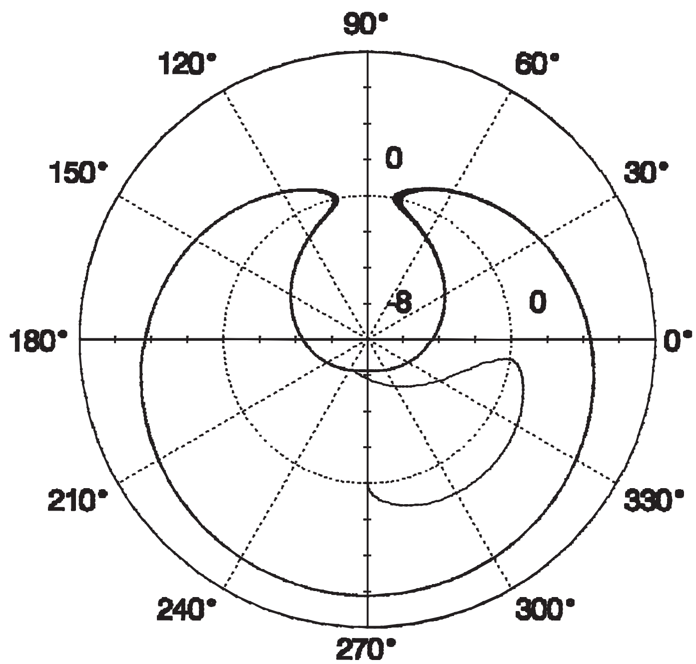

Figure 7.2: Reflexive stabilizer control of pendulum (μ=0.2). (velocity on radial axis in radians per sec)

pendulum starts at 270 degrees with zero velocity. The tail in the center of Figure 7.2 is the swing up to the limit cycle. The final limit cycle flips the pendulum back and forth. At the top of each swing, the pendulum is in the control domain of the neural network.

7.2.2 Reflexive Stabilizer for Stability Assurance

The next test examines the ability of the reflexive stabilizer to prevent failure of the cart/pole system. This case is difficult to prevent from failure since the system is inherently unstable and has a single control actuator for a system with two degrees of freedom. The design of the cart/pole reflexive stabilizer is presented in Section 5.3.

Functionally, the cart/pole reflexive stabilizer performed flawlessly. In every test conducted, the pole was never dropped and the cart was maintained within 8 meters of the center. The side to side motion of the cart was greater than the 2.4 meters of the Barto *et al.*, (1983) studies, but this is caused by the larger pole angles increasing the nonlinearity of system.

The reflexive stabilizer can maintain stability even under heavy disturbances. Some tests were conducted with random pole position disturbances with a range of 5 degrees; no failures occurred. Overall, the cart/pole system was simulated in various tests for over 6 million seconds

(~70 days) of simulated time without failure. For stability assurance, the reflexive stabilizer was highly successful. The next section, describes the tests in which the knowledge contained in the reflexive stabilizer was used to train the neural network controller.

7.3 Learning via Distal Error

A primary motivation for the development of the new distal-error training algorithm was to allow the reflexive stabilizer to train the network. For details of the distal-error training algorithm, see Section 5.5. When the reflexive stabilizer is used for training, the control architecture can be simplified to only the neural network and the reflexive stabilizer. This is illustrated in Figure 7.3.

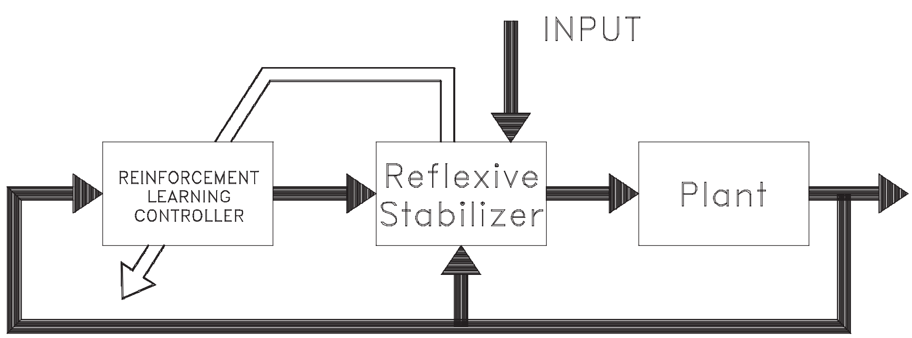

Figure 7.3: Training a neural network with a reflexive stabilizer.

For this type of training to be effective, the requisite control actions just prior to encountering the reflexive stabilizer must be in the same direction as the reflexive stabilizer corrective forces. The cart/pole reflexive stabilizer meets this criterion, since actions near the reflexive stabilizer boundary should be in the same direction as the reflexive stabilizer forces. In contrast, the underpowered pendulum does not meet this criterion. The reflexive stabilizer adds energy to the pendulum while the neural network controller needs to remove and add energy from the system. Thus, control actions inferred inward from the pendulum reflexive stabilizer boundaries are inappropriate.

7.3.1 Training Tests

To demonstrate the distal-error training algorithm, the cart/pole system was used. The reflexive stabilizer is the same as presented in Section 5.3. No reflexive stabilizer changes are needed for the distal-error training algorithm. The neural network is set to learn only when there is a transition of control from the neural network to reflexive stabilizer. In this configuration, the neural network is trained by punishment.

No training occurred while the system was under reflexive control. At first, it would appear that the network would learn faster if training took place both at the transition and during operation of the reflexive stabilizer. The neural network, however, is a gaussian learner, so the network learns to act like the reflexive stabilizer rather than learning the desired inferences.

Figure 7.4 shows an example of training the neural network controller with a reflexive stabilizer. The learning curve is peculiar since it does not

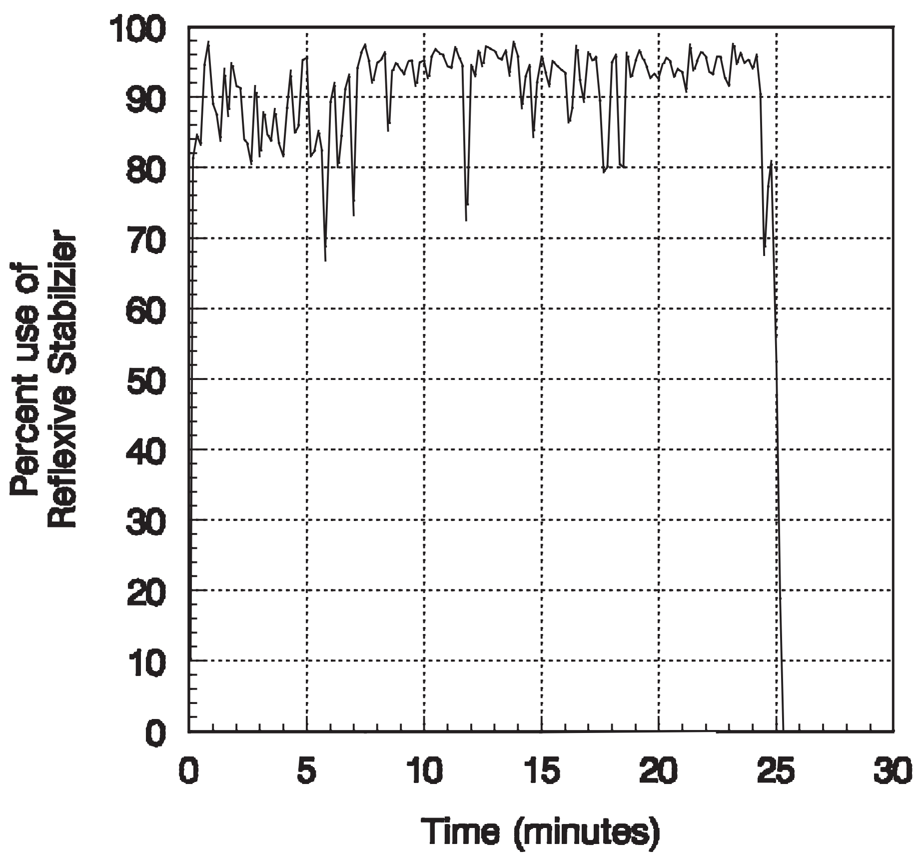

Figure 7.4: Test of training a neural network with a reflexive stabilizer for the cart/pole problem.

show a gradual improvement in performance. The neural network controller performs poorly and then suddenly takes over control. Even after close inspection (See Figure 7.5) of the transition from reflexive stabilizer to neural network control, the learning curve is still unusual. The shape of the first

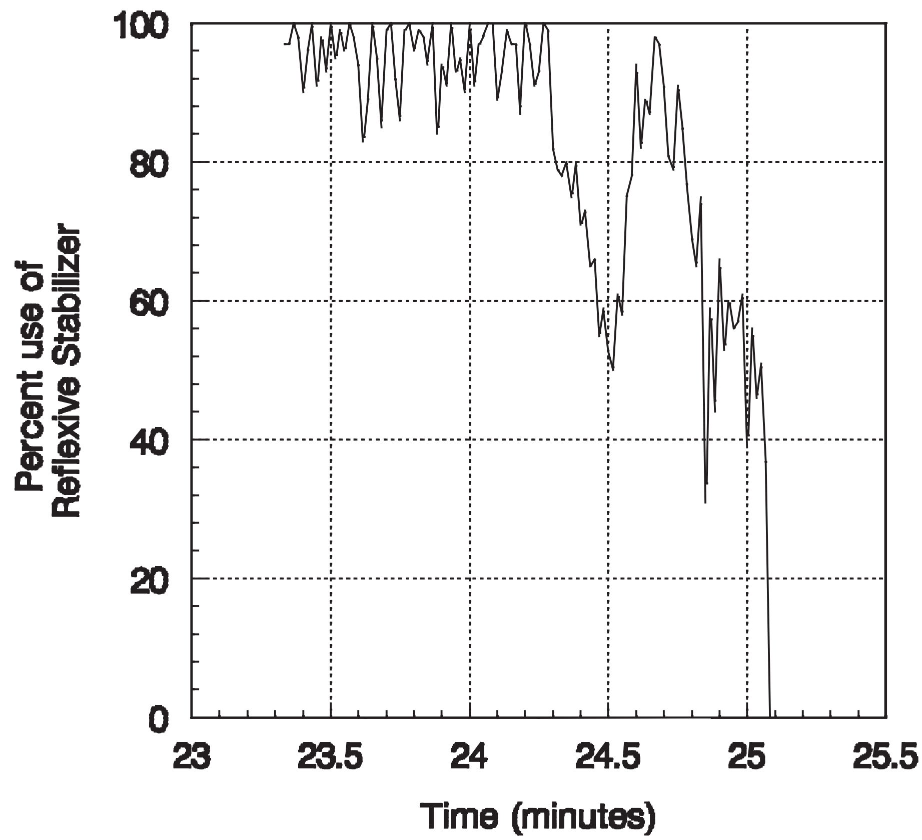

Figure 7.5: Transition region of Figure 7.4.

part of the curve, up to 24 minutes, is common in neural network training. During this period the network is adjusting the hidden nodes to correct orientations for solving the problem. After 24 minutes, the hidden units have the correct basic representations and the network can solve the problem. The final portion of the learning curve, however, is not monotonic. This probably is the result of the hidden units making a final adjustment, but the distal-error training or the on-line training may be the cause. To explain this unusual learning, the differences between standard back-propagation training and the distal-error algorithm were considered.

The distal-error training algorithm and standard back-propagation training are obviously different. Back-propagation training has a target output, but the distal-error algorithm infers the target from the reflexive stabilizer boundaries. The on-line training also adds the effects of continuously changing training examples and stochastic approximation of the neural network control surface. In contrast, back-propagation networks are normally trained with repeated presentations of a fixed data set. Because of the large differences in the training, there is no immediate explanation for the unusual training. Therefore, since a theoretical explanation is not available, the learning was explored via simulation.

To explore the unusual learning, two extended simulations were conducted. The first test explored the effect of regularly forcing the cart/pole system into the reflexive stabilizer boundaries. Each time the neural network